Django backend

Since the system is relatively complex, we will start from the first part and focus on the architecture. The simplest part: **the models.**

Recall the previous discussion on models, where the main task is to create and complete two different classes: Sensors and Measurements.

Different data types of models:

#Integer	age = models.IntegerField()
#Floating point	price = models.FloatField()
#Positive integer	quantity = models.PositiveIntegerField()
#Maximum length, string	name = models.CharField(max_length=100)
#Large text field	description = models.TextField()
#Boolean value field	is_active = models.BooleanField(default=True)
#Date field	birth_date = models.DateField()
#Time field	start_time = models.TimeField()
#Email field	email = models.EmailField()
#File upload field	file = models.FileField(upload_to = 'uploads/')
#Image upload field	avatar = models.ImageField(upload_to = 'avatars/')
#URL field	website = models.URLField()

Sensor_Project/ models.py

```
from django.db import models

# Create your models here.
class Sensors(models.Model): # Define the Sensors model
    Sensor_ID = models.CharField(max_length=100)
    Location = models.CharField(max_length=100)
    Sensor_Key = models.PositiveIntegerField()

    class Meta:  # Set the model's metadata
        db_table = 'Sensors' # Specify the database table name as 'sensors'
```

```
class Measurements(models.Model): # Define the Measurements model
    ID = models.PositiveIntegerField()
    Temperature = models.FloatField()
    Humidity = models.FloatField()
    Timestamp = models.DateTimeField()
    TimeZone = models.CharField(max_length=100, default='unknown')
    Temperature_Change_Rate = models.CharField(max_length=50, null=True,
blank=True)
    Humidity_Change_Rate = models.CharField(max_length=50, null=True, blank=True)
    Comfort_Index = models.FloatField()
    Location = models.CharField(max_length=100)
    Sensor_ID = models.PositiveIntegerField()
    Comfort_Level = models.CharField(max_length=100,
default='unknown')

    class Meta: # Set the model's metadata
        db_table = 'Measurements' # Specify the database table name as
'measurements'
```

Django Migrations

Migrations are a feature of Django used to synchronize models with the database schema. This can include creating, modifying, or deleting database tables and fields. Django uses migration files to track changes in the database schema. Whenever there are changes to the models (such as adding, deleting, or modifying fields), a migration file needs to be created and then applied to update the database structure.

Migration Commands:

- **makemigrations:** This command generates new migration files based on changes to the models.
- **migrate:** This command applies the changes from the migration files to the database.

- **python manage.py makemigrations myapp** **# Generate migration files**
- **python manage.py migrate** **# Apply migration files to the database**